# Rage Against The Virtual Machine: Hindering Dynamic Analysis of Android Malware

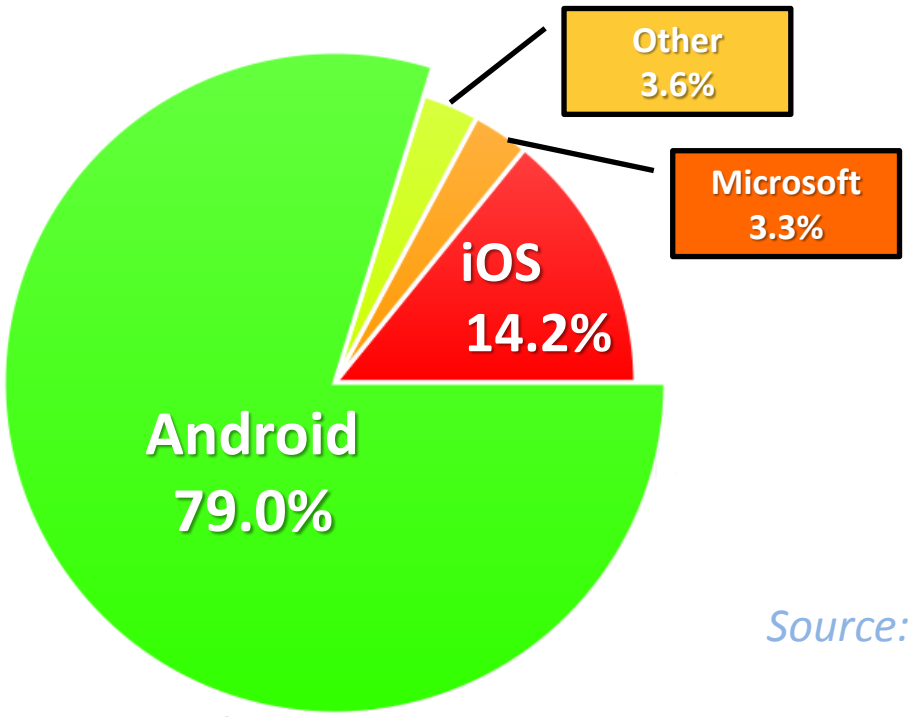
**Thanasis Petsas**, Giannis Voyatzis,
Elias Athanasopoulos, Sotiris Ioannidis,


Michalis Polychronakis

# Android Dominates Market Share

- Smartphones have overtaken client PCs
- Android accounted for 79% of global smartphone market in 2013

**Q2 2013 Smartphone Market Share**

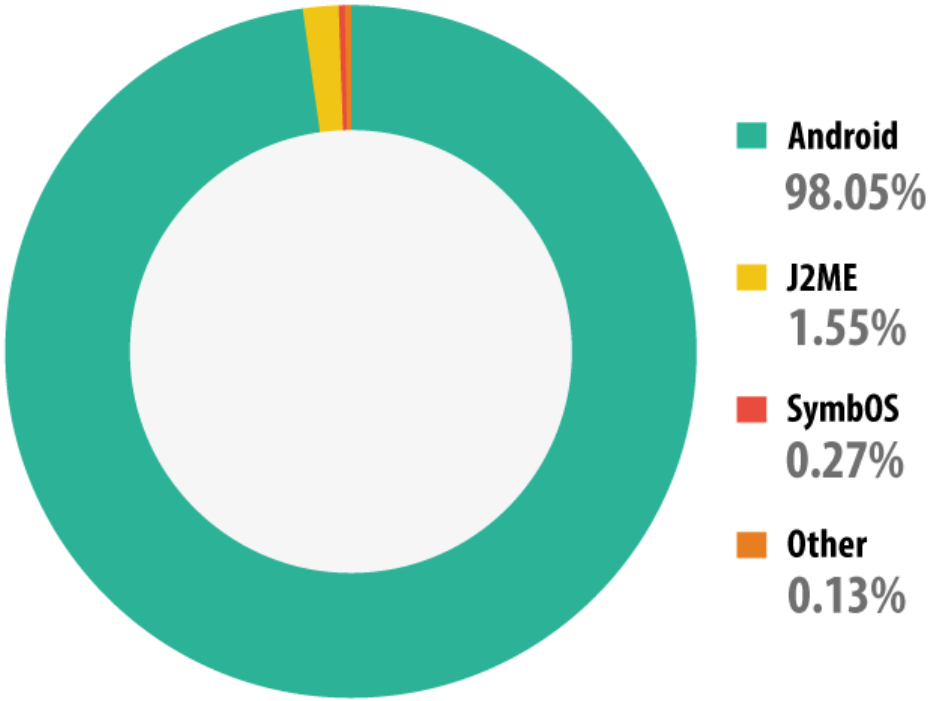



Source: Gartner

# Android Malware

- 98% of all mobile threats target Android devices

Source: KASPERSKY lab




**Distribution of mobile malware detected by platform – 2013**

# Android specific anti-malware tools

- Static analysis tools (AV apps)

- Dynamic analysis services

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures

- Dynamic analysis services

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users

- Dynamic analysis services

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection

- Dynamic analysis services

VisualThreat

Tracedroid

apkscan

SandDroid

CopperDroid

FORESAFE MOBILE SECURITY

MS MOBILE SANDBOX

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - ***How to evade static analysis?***

- Dynamic analysis services

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - ***How to evade static analysis?***


- Dynamic analysis services

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - ***How to evade static analysis?***


- Dynamic analysis services
  - Used by security companies

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - ***How to evade static analysis?***


- Dynamic analysis services
  - Used by security companies
  - Run applications on an Emulator

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - ***How to evade static analysis?***


- Dynamic analysis services
  - Used by security companies
  - Run applications on an Emulator
  - Detect *suspicious* behavior

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - ***How to evade static analysis?***


- Dynamic analysis services
  - Used by security companies
  - Run applications on an Emulator
  - Detect *suspicious* behavior
  - ***How to evade dynamic analysis?***

# Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - ***How to evade static analysis?***


- Dynamic analysis services
  - Used by security companies
  - Run applications on an Emulator
  - Detect *suspicious* behavior
  - ***How to evade dynamic analysis?***

# Our Study

**Objective:** *Can we effectively detect Android emulated analysis environment?*

- A taxonomy of emulation evasion heuristics
- Evaluation of our heuristics on popular dynamic analysis services for Android
- Countermeasures

# VM Evasion Heuristics

| Category | Type | Examples |
|---|---|---|
| **Static** | Pre-installed static information | IMEI has a fixed value |
| **Dynamic** | Dynamic information does not change | Sensors produce always the same value |
| **Hypervisor** | VM instruction emulation | Native code runs differently |

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

| ***IMEI*** | *123456789012347* | *null* |
|---|---|---|

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI

**Android Pincer malware family**

| ***IMEI*** | `123456789012347` | `null` |
|---|---|---|

- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`

- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

**Android Pincer malware family**

| ***IMEI*** | *123456789012347* | *null* |
|---|---|---|

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

**Android Pincer malware family**

| | | |
|---|---|---|
| ***IMEI*** | *123456789012347* | *null* |
| ***MODEL*** | *Nexus 5* | *google_sdk* |

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

**Android Pincer malware family**

| | | |
|---|---|---|
| ***IMEI*** | *123456789012347* | *null* |
| ***MODEL*** | *Nexus 5* | *google_sdk* |

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

**Android Pincer malware family**

| | | |
|---|---|---|
| ***IMEI*** | *12345678901234*7 | *null* |
| ***MODEL*** | *Nexus 5* | *google_sdk* |
| **/proc/net/tcp** | *Ordinary network* | *Emulated network* |

# Static Heuristics

- Device ID (***IdH***)
  - IMEI, IMSI
- Current build (***buildH***)
  - Fields: `PRODUCT`, `MODEL`, `HARDWARE`
- Routing table (***netH***)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

**Android Pincer malware family**

| | | |
|---|---|---|
| ***IMEI*** | *123456789012347* | *null* |
| ***MODEL*** | *Nexus 5* | *google_sdk* |
| **`/proc/net/tcp`** | *Ordinary network* | *Emulated network* |

# Dynamic Heuristics (1/3)

**GPS**

**Accelerometer** **Gyroscope**

**Gravity Sensor** **Proximity Sensor**

**Rotation Vector** **Magnetic Field**


Sensors:

- A key difference between mobile & conventional systems
- new opportunities for mobile devices identification
- ***Can emulators realistically simulate device sensors?***

# Dynamic Heuristics (1/3)

**GPS**

**Accelerometer** **Gyroscope**

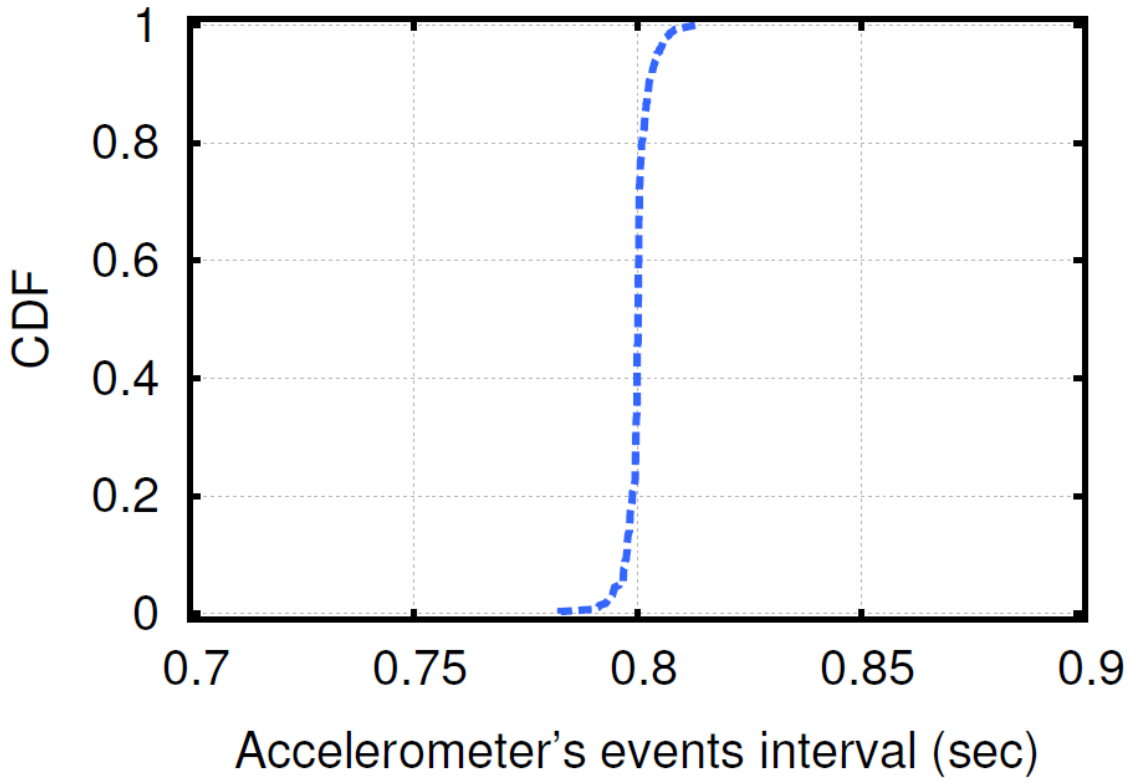
**Gravity Sensor** **Proximity Sensor**

**Rotation Vector** **Magnetic Field**

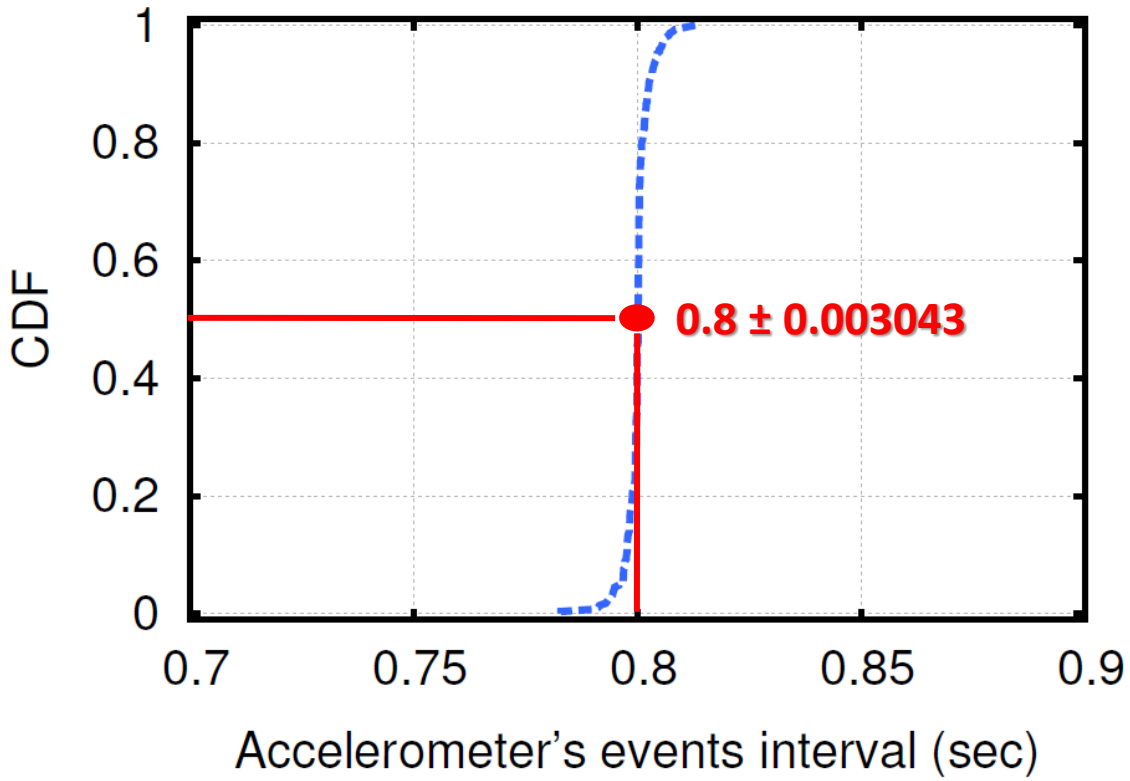

Sensors:

- A key difference between mobile & conventional systems
- new opportunities for mobile devices identification
- ***Can emulators realistically simulate device sensors?***
  - Partially: same value, equal time intervals

# Dynamic Heuristics (2/3)



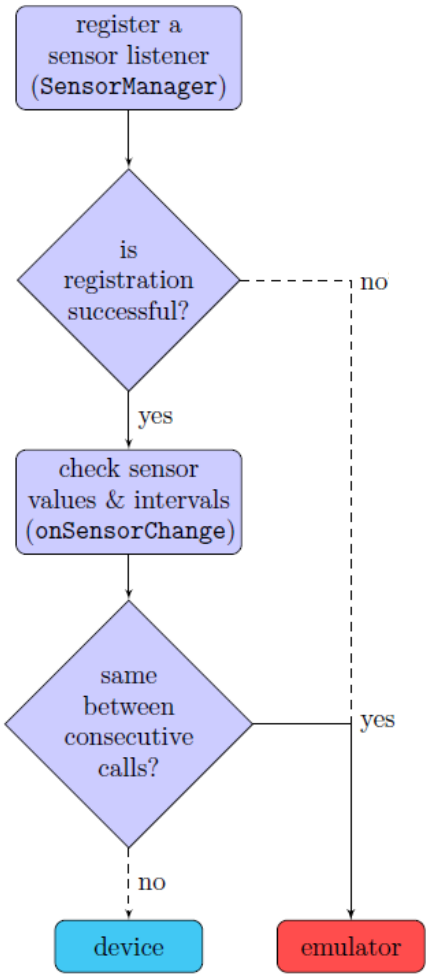
# Dynamic Heuristics (2/3)




Generation of the same value at equal time intervals

# Dynamic Heuristics (3/3)

- Sensor-based heuristics
- Android Activity that monitors sensors' output values
- We implemented this algorithm for a variety of sensors
  - Accelerometer (**accelH**)
  - magnetic field (**magnFH**)
  - rotation vector (**rotVecH**),
  - proximity (**proximH**)
  - gyroscope (**gyrosH**)



# Hypervisor Heuristics

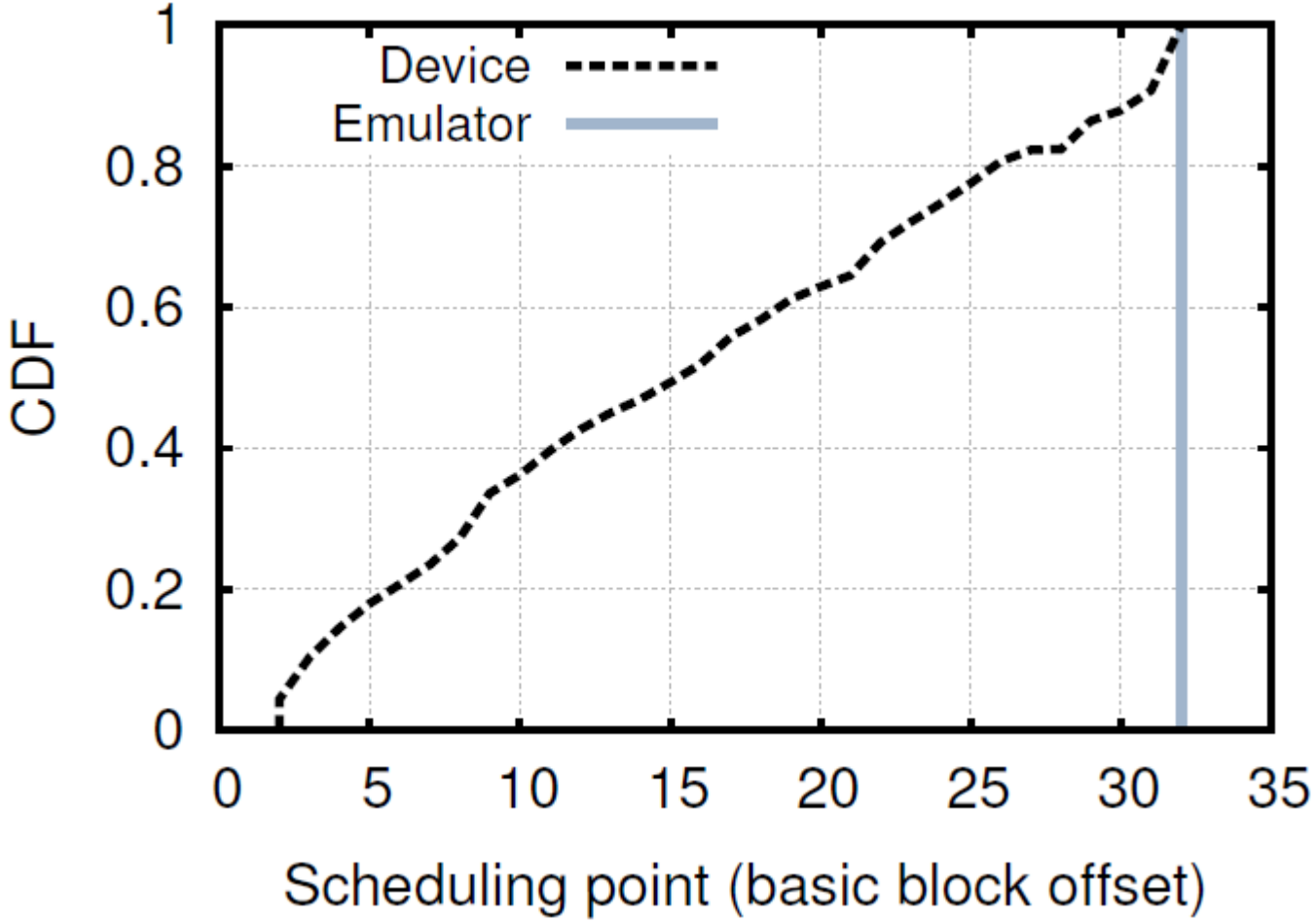
- Try to identify the hosted virtual machine
- Android Emulator is based on **QEMU**
- Our heuristics
  - Based on QEMU's incomplete emulation of the actual hardware
  - Identify QEMU scheduling
  - Identify QEMU execution using self-modifying code

# Identify QEMU Scheduling (1/2)

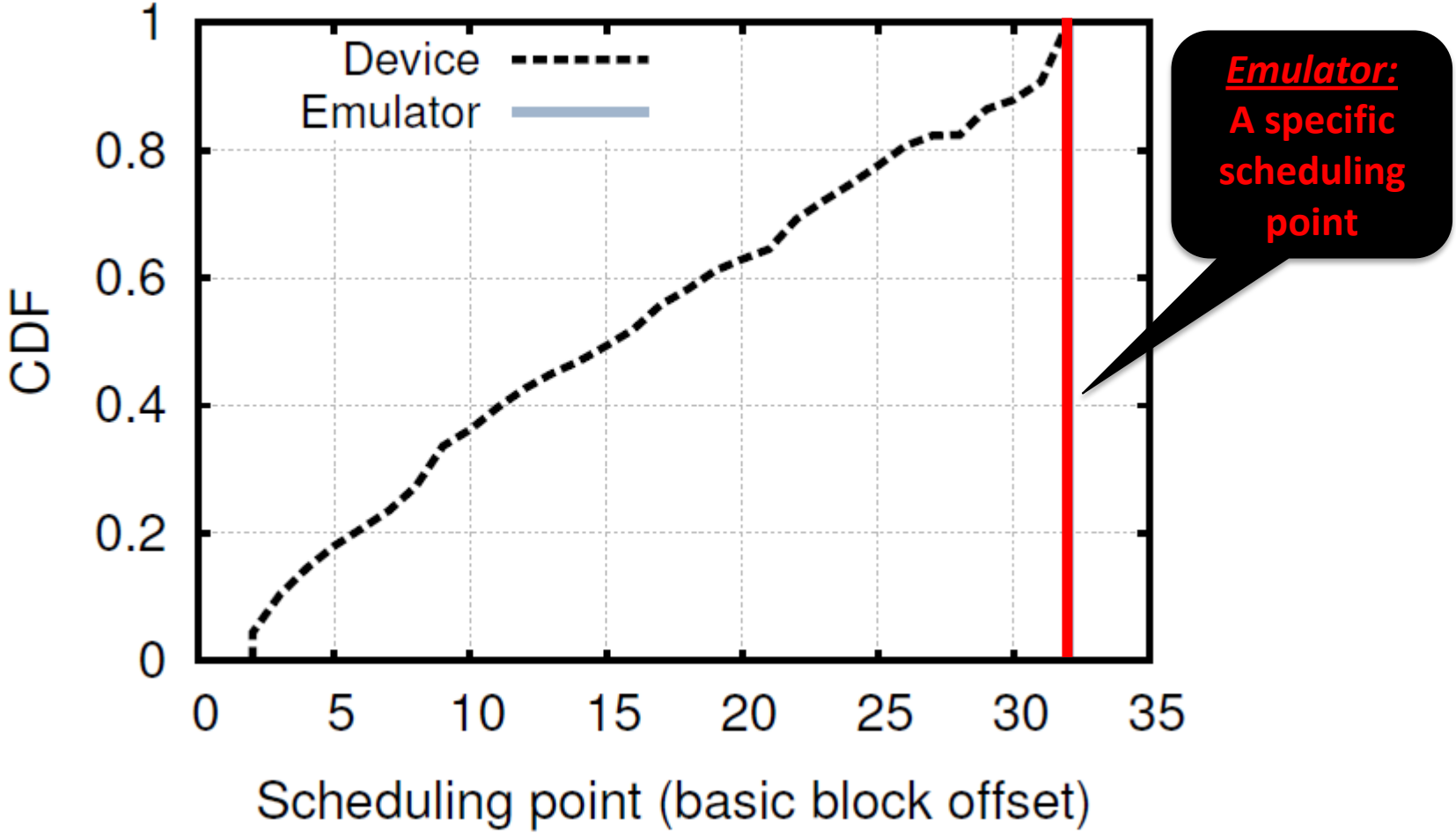
- Virtual PC in QEMU
  - is updated only after the execution of a basic block (**`branch`**)
  - OS scheduling does not occur during a basic block

- QEMU Binary Translation (BT) Detection DEXLabs
  - Monitor scheduling addresses of a thread
    - Real Device: Various scheduling points
    - Emulator: A unique scheduling point
  - **BTdetectH**

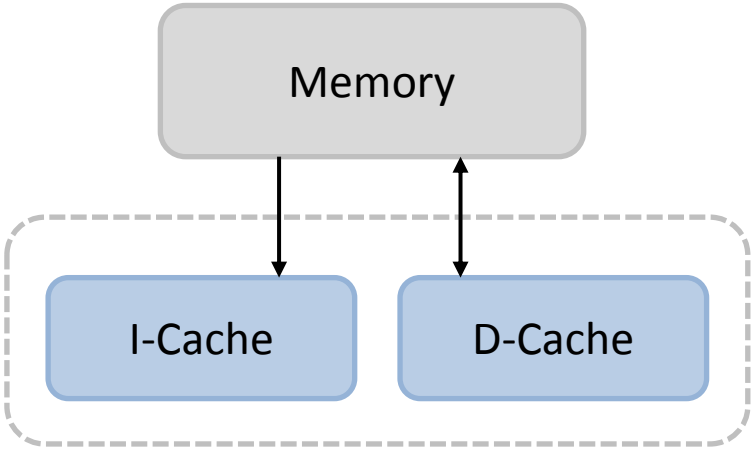
# Identify QEMU Scheduling (2/2)



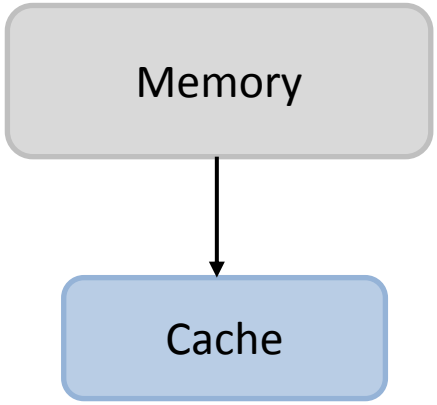
# Identify QEMU Scheduling (2/2)



# ARM Architecture

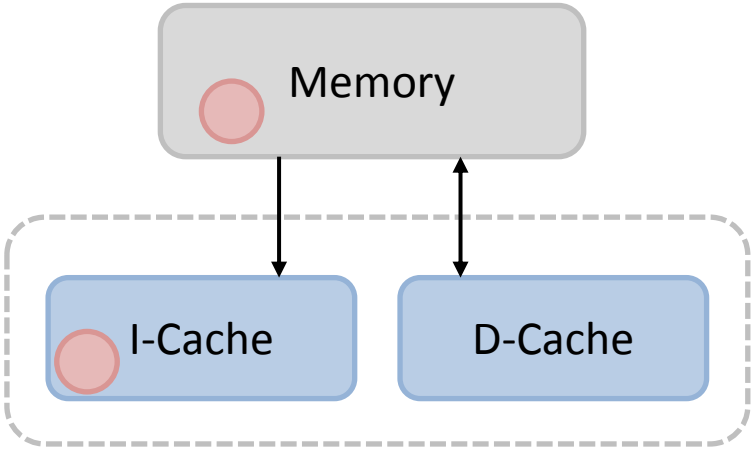



**Device**

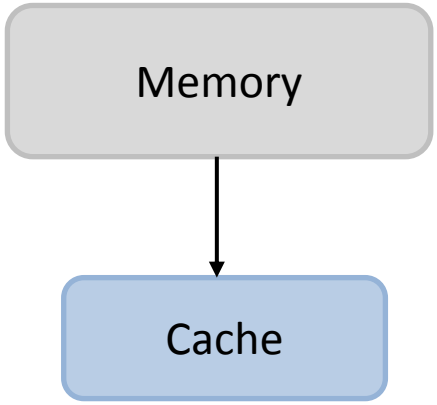



**Emulator**

# ARM Architecture



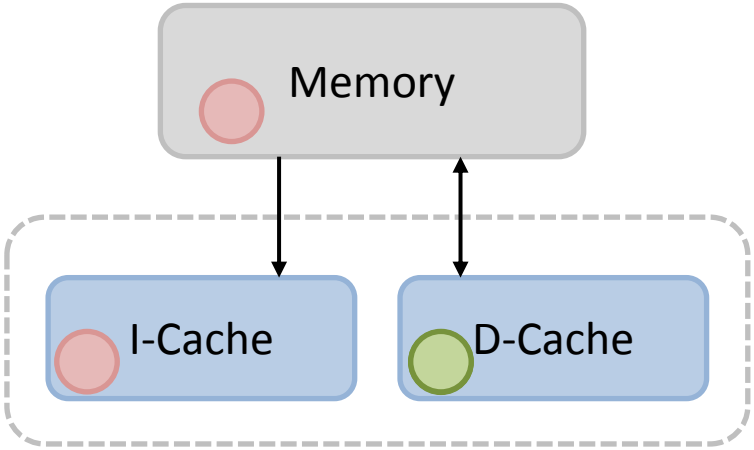

**Device**



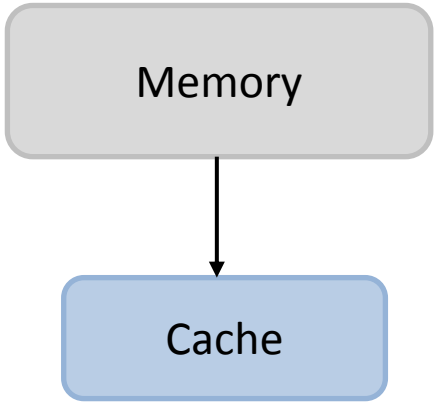

**Emulator**

old code

# ARM Architecture




**Device**

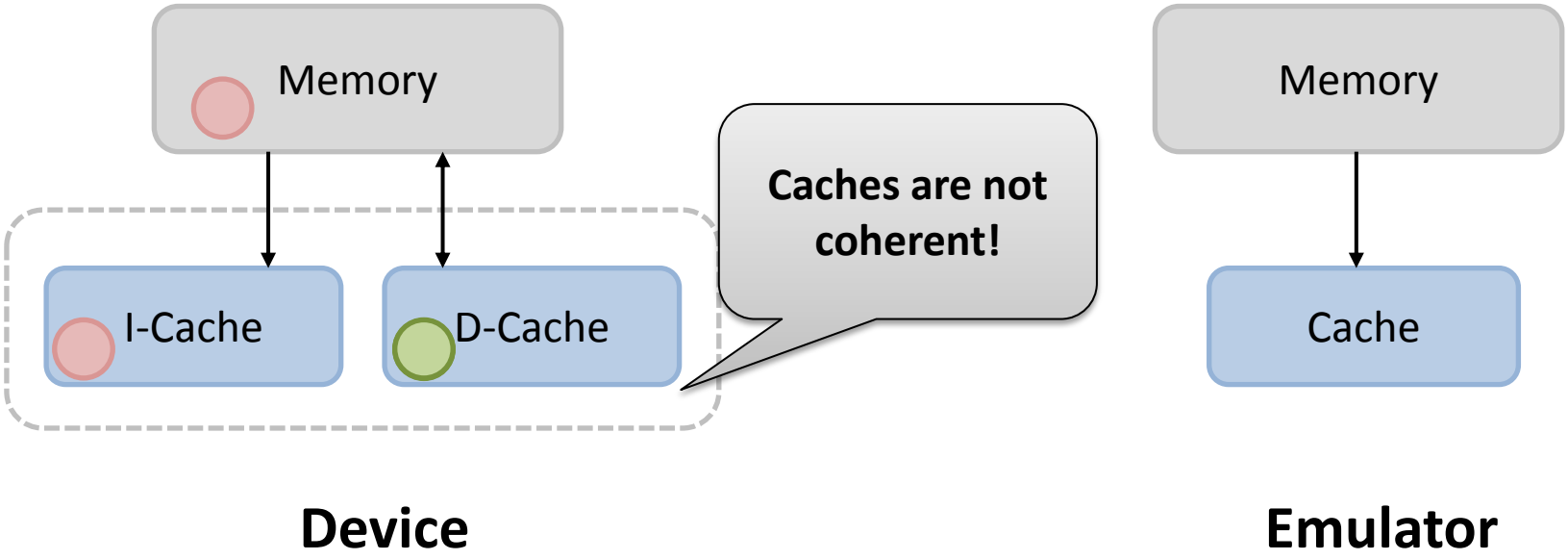



**Emulator**

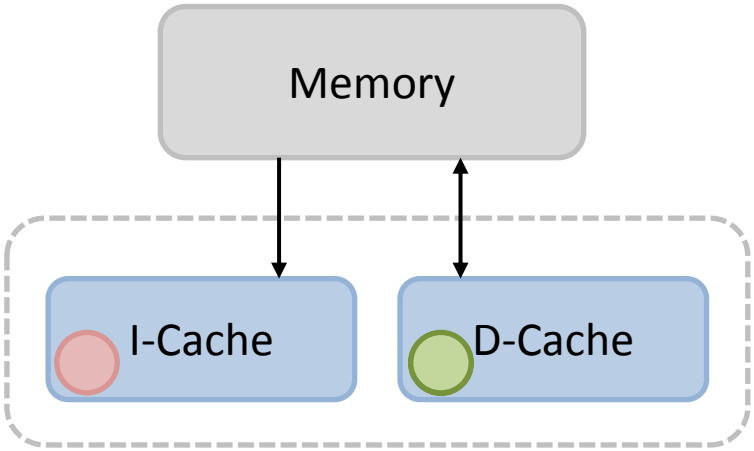
old code

new code

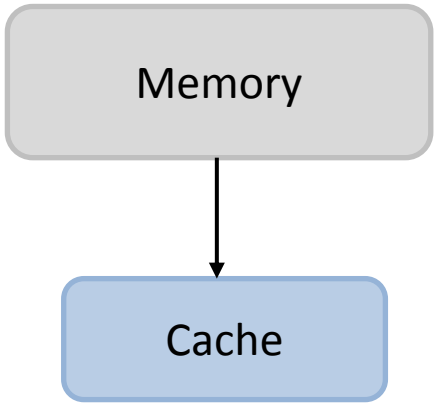
# ARM Architecture



# ARM Architecture




**Device**

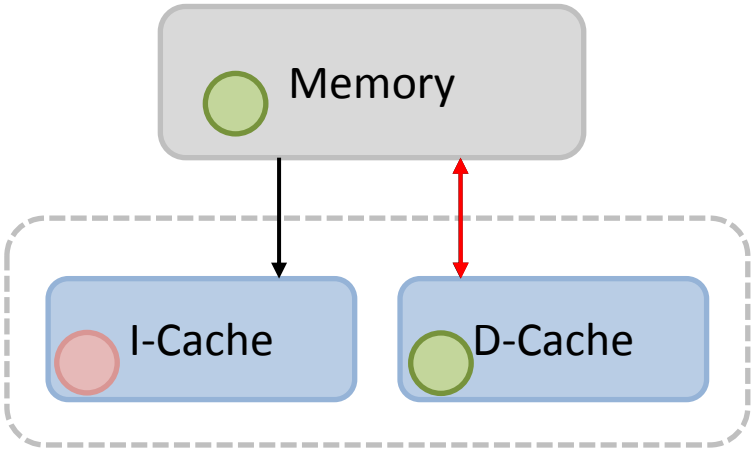



**Emulator**

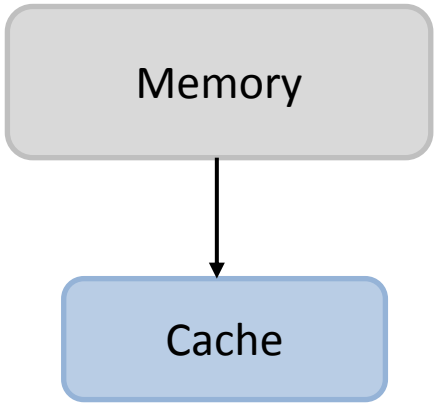
old code

new code

# ARM Architecture




**Device**



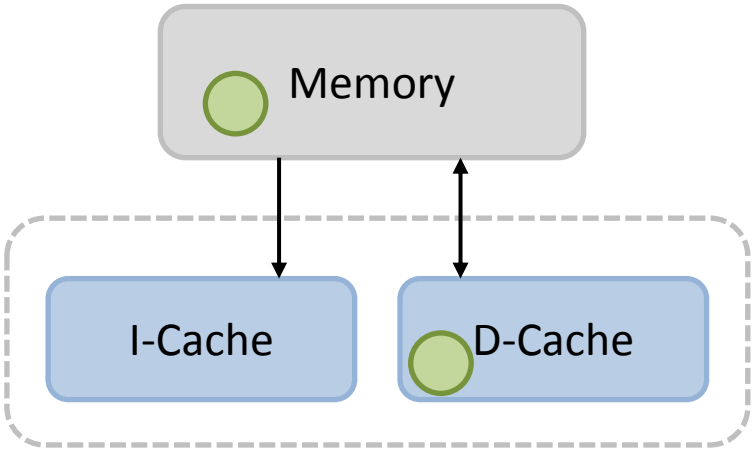

**Emulator**

old code

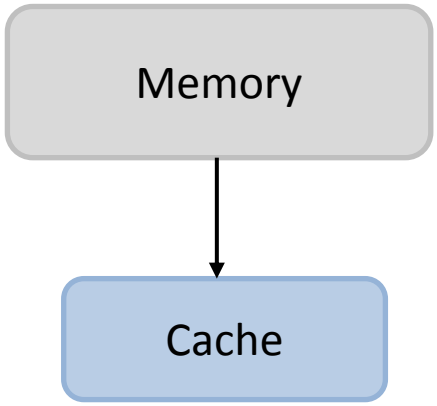
new code

**Clean the D-Cache range**

# ARM Architecture




**Device**



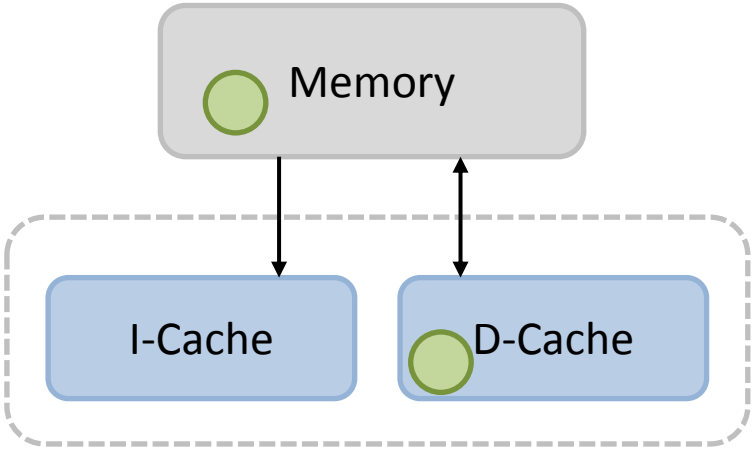

**Emulator**

old code

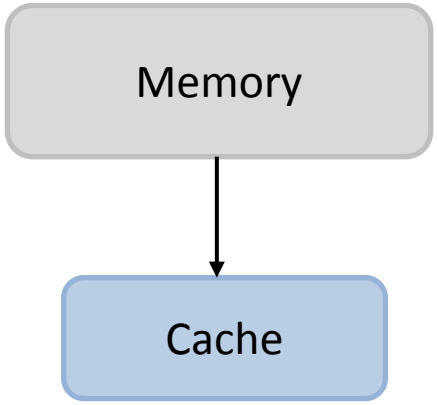
new code

**Invalidate the I-Cache**

# ARM Architecture




**Device**

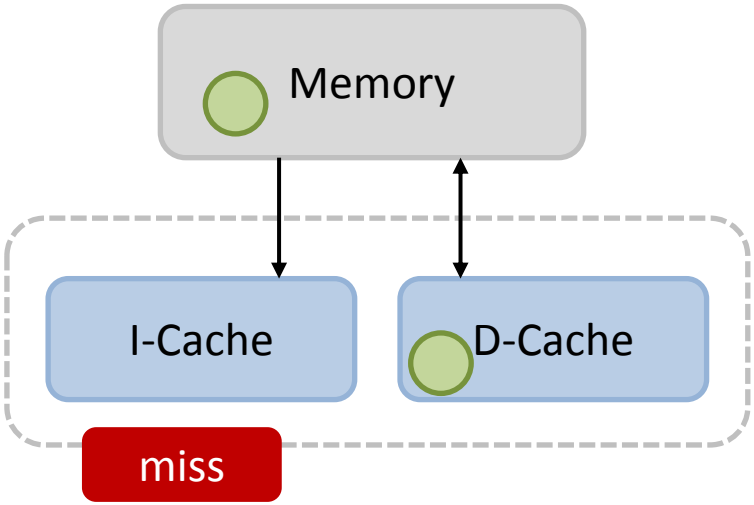



**Emulator**

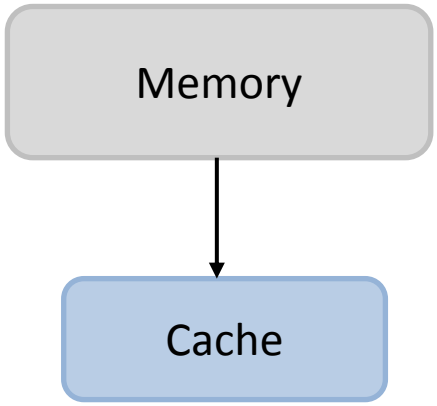
old code

new code

# ARM Architecture




**Device**



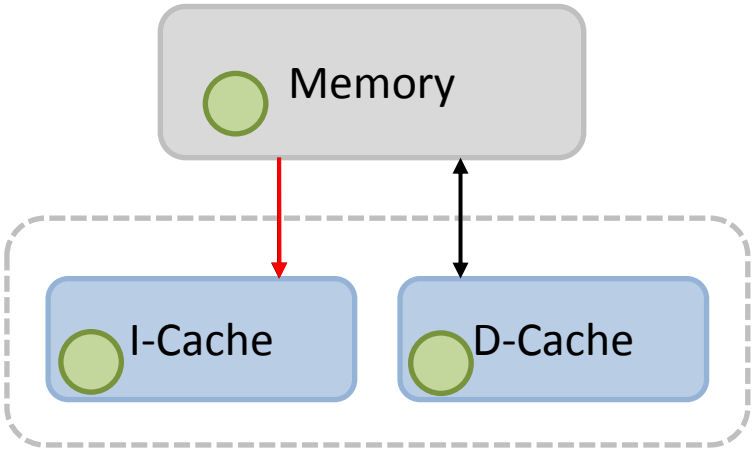

**Emulator**

old code

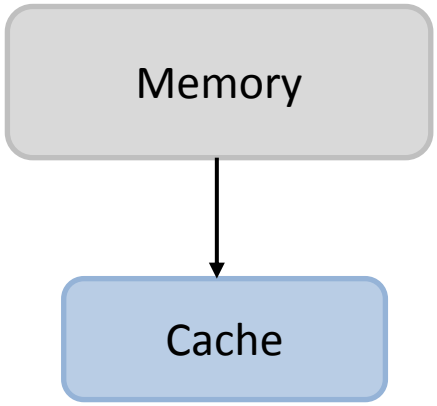
new code

**Run the code**

# ARM Architecture




**Device**



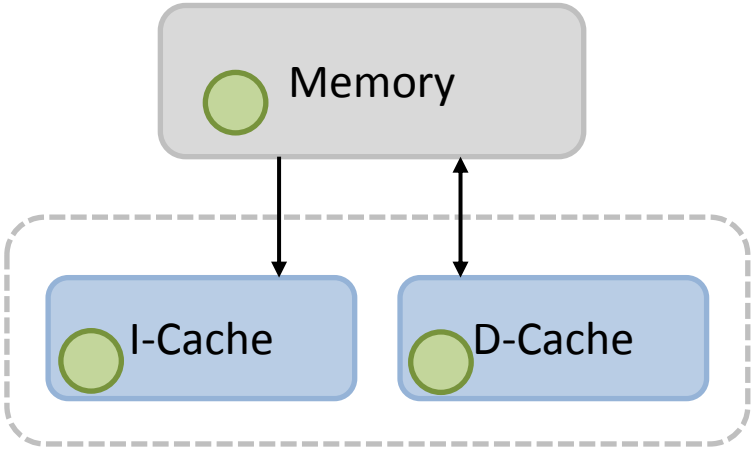

**Emulator**

old code

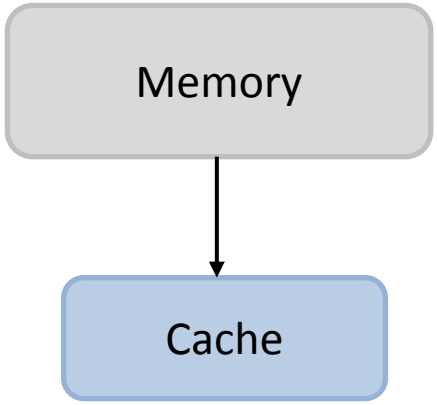
new code

**Run the code**

# ARM Architecture




**Device**




**Emulator**

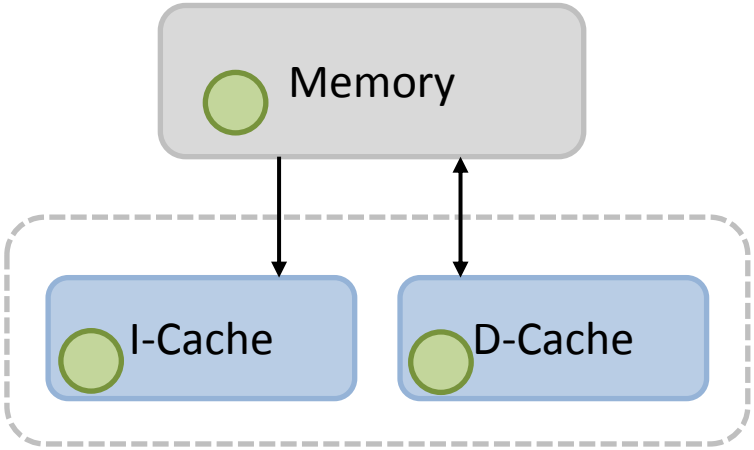
old code

new code

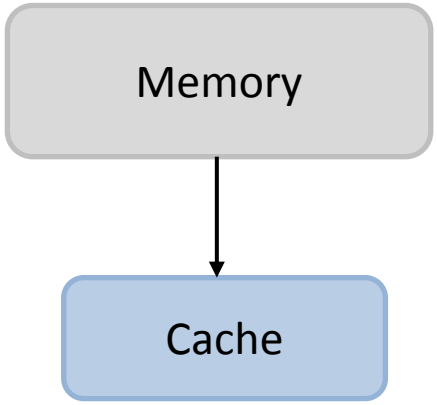
**Android `cacheflush`:**

1. **Clean the D-Cache range**
2. **Invalidate the I-Cache**

# ARM Architecture




**Device**




**Emulator**

old code

new code

# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();
   patch_code(&swap, &patch, &f2);

   code_func();
}
```

# Identify QEMU execution – **xFlowH**

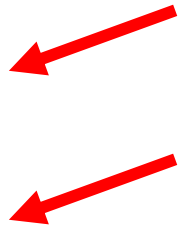
```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();

   patch_code(&swap, &patch, &f2);

   code_func();
}
```

# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();

   patch_code(&swap, &patch, &f2);

   code_func();
}
```

# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();
   patch_code(&swap, &patch, &f2);

   code_func();
}
```



# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();

   patch_code(&swap, &patch, &f2);

   code_func();
}
```

# Identify QEMU execution – **xFlowH**

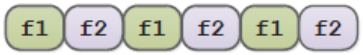
```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);
   cacheflush();
   code_func();
   patch_code(&swap, &patch, &f2);
   cacheflush();
   code_func();
}
```

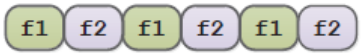
**with cacheflush:**

# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);
   cacheflush();
   code_func();
   patch_code(&swap, &patch, &f2);
   cacheflush();
   code_func();
}
```

**with cacheflush:**

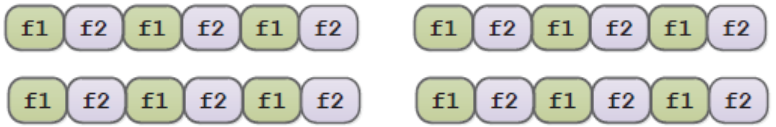


# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);
   cacheflush();
   code_func();
   patch_code(&swap, &patch, &f2);
   cacheflush();
   code_func();
}
```

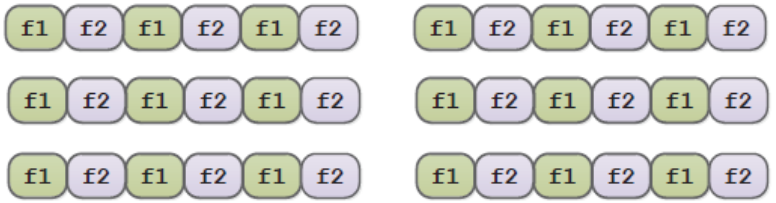
**with cacheflush:**



# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);
   cacheflush();
   code_func();
   patch_code(&swap, &patch, &f2);
   cacheflush();
   code_func();
}
```

**with cacheflush:**

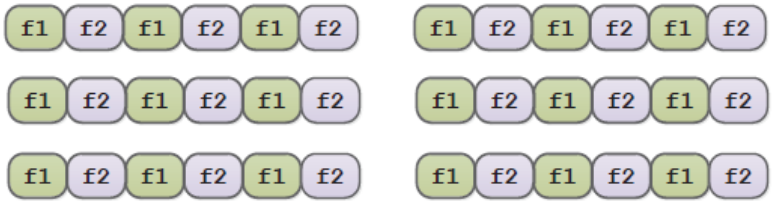


# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);
   cacheflush();
   code_func();
   patch_code(&swap, &patch, &f2);
   cacheflush();
   code_func();
}
```

**with cacheflush:**

same behavior.

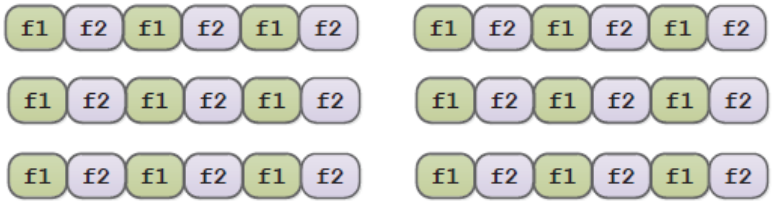


# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();
   patch_code(&swap, &patch, &f2);

   code_func();
}
```

**with cacheflush:**

same behavior.



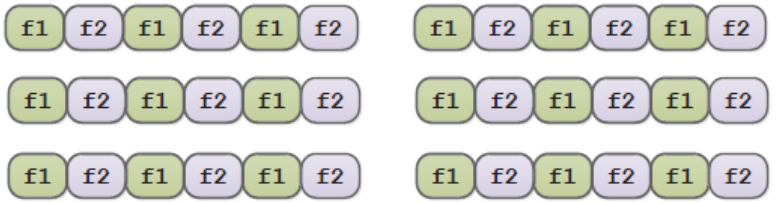

**without cacheflush:**

# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();
   patch_code(&swap, &patch, &f2);

   code_func();
}
```

**with cacheflush:**

same behavior.



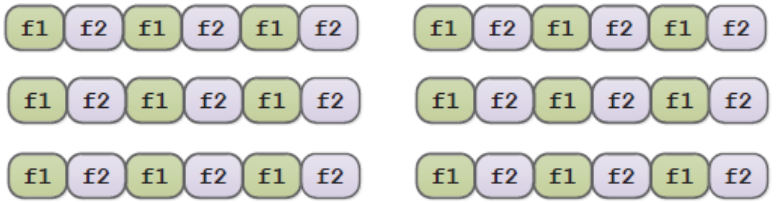

**without cacheflush:**

# Identify QEMU execution – **xFlowH**

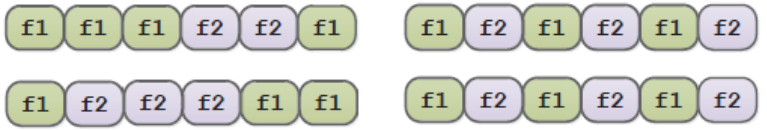
```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();
   patch_code(&swap, &patch, &f2);

   code_func();
}
```

**with cacheflush:**

same behavior.




**without cacheflush:**

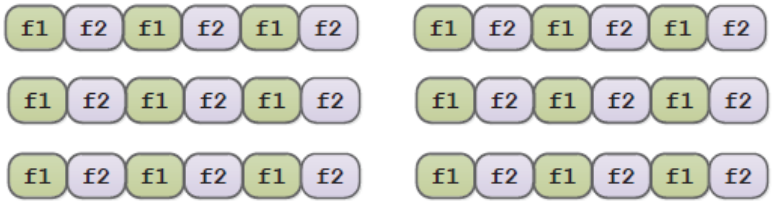


# Identify QEMU execution – **xFlowH**

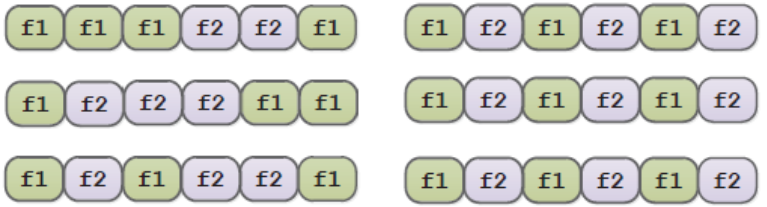
```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();
   patch_code(&swap, &patch, &f2);

   code_func();
}
```

**with cacheflush:**

same behavior.




**without cacheflush:**

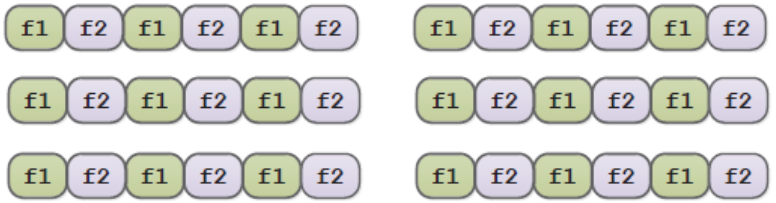


# Identify QEMU execution – **xFlowH**

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
   NULL,
   16 * 4,
   PROT_READ | PROT_WRITE | PROT_EXEC,
   MAP_PRIVATE | MAP_ANONYMOUS,
   -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,

for (i=0; i<N; i++) {
   patch_code(&swap, &patch, &f1);

   code_func();
   patch_code(&swap, &patch, &f2)

   code_func();
}
```

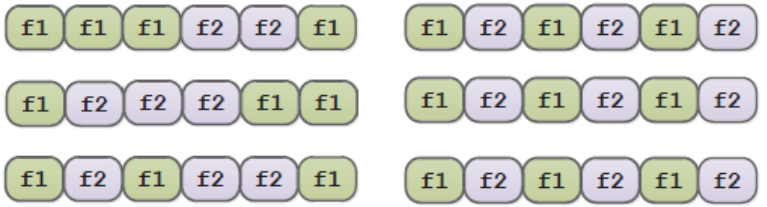
**with cacheflush:**

**same behavior.**




**without cacheflush:**

**different behavior!**



# Implementation

- Use of Android SDK for static & dynamic heuristics
- Use of Android NDK for hypervisor heuristics
- Implementation of an Android app
  - runs the heuristics
  - send the results to an HTTP server
- Repackaging of well known Android malware samples
  - Smali/Baksmali
  - Apktool
  - Patching the Smali Dalvik Bytecode

# Evaluation: Malware Set

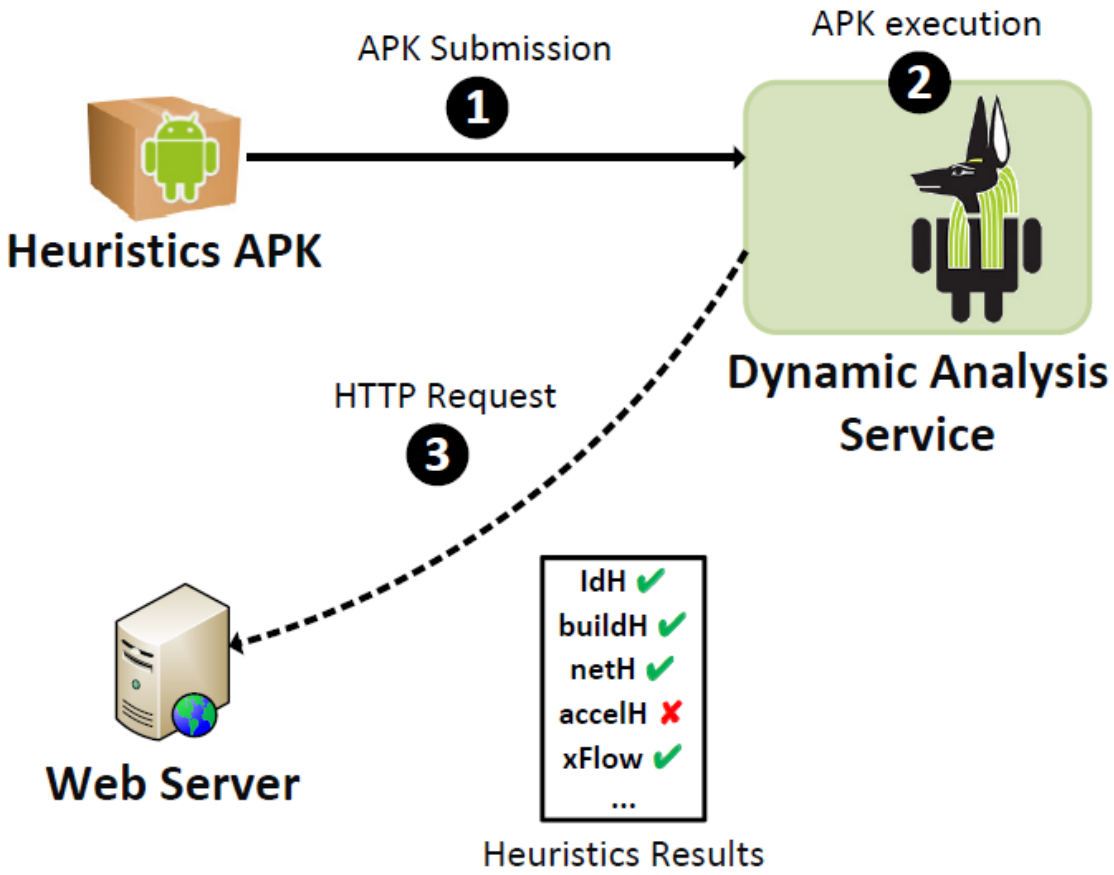
| Family | Package name | Heuristic | Description |
|---|---|---|---|
| BadNews | ru.blogspot. playsib.savageknife | magnFH | Data extrusion |
| BaseBridge | com.keji.unclear | accelH | Root exploit |
| Bgserv | com.android. vending.sectool.v1 | netH | Bot activity |
| DroidDream | com.droiddream. bowlingtime | gyrosH | Root expoit |
| DroidKungFu | com.atools.cuttherope | rotVecH | Root exploit |
| FakeSMS Installer | net.mwkekdsf | proximH | SMS trojan |
| Geinimi | com.sgg.sp | buildH | Bot avtivity |
| Zsone | com.mj.iCalendar | idH | SMS trojan |
| JiFake | android.packageinstaller | BTdetectH | SMS trojan |
| Fakemart | com.android.blackmarket | xFlowH | SMS trojan |

Source: http://contagiominidump.blogspot.com

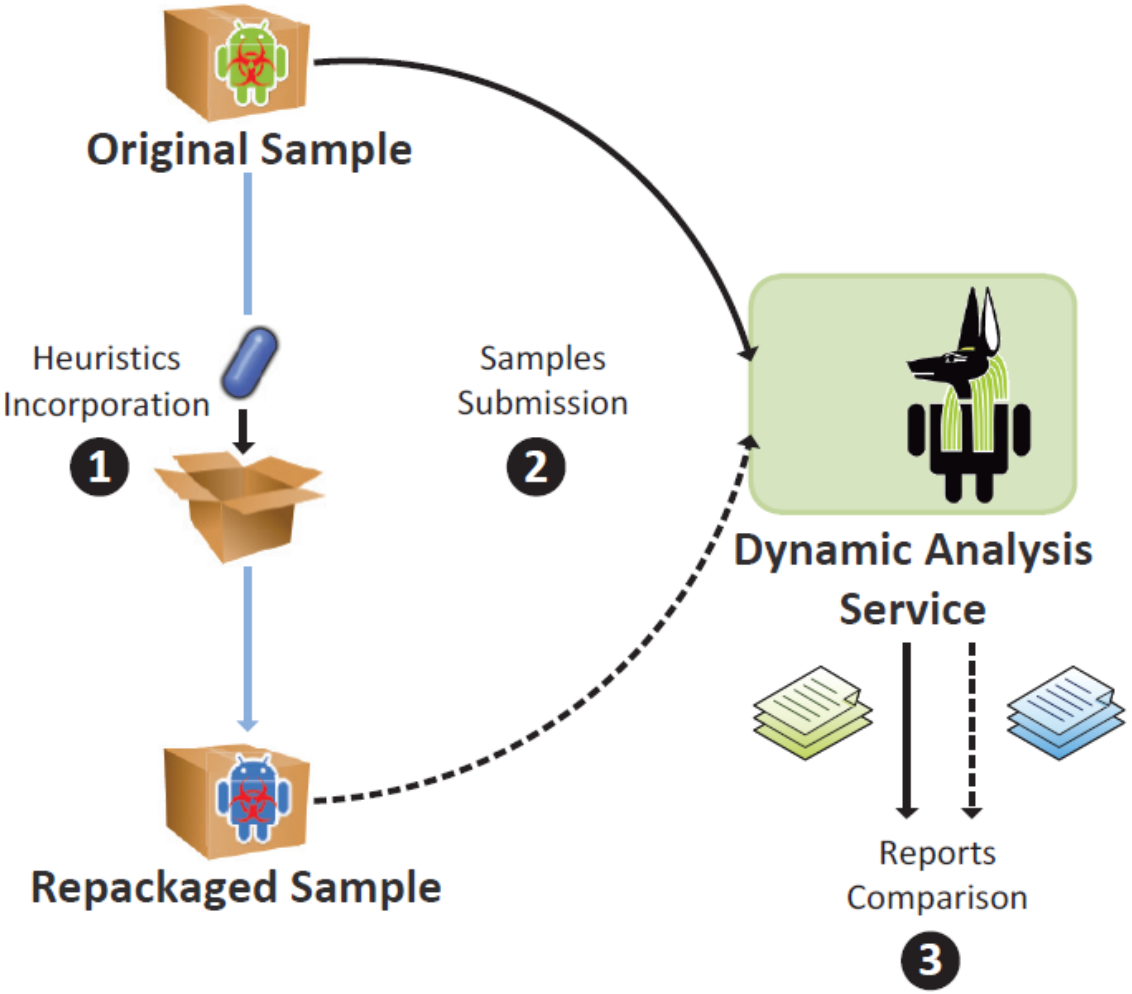
# Evaluation: Dynamic Analysis Services

- Stand alone tools
  - DroidBox, DroidScope, TaintDroid
- Online services
  - Andrubis, SandDroid, ApkScan, Visual Threat, TraceDroid, CopperDroid, APK Analyzer, ForeSafe, Mobile SandBox

# Methodology (1/2)



# Methodology (2/2)



# Resilience of dynamic analysis tools

| | Static | | | Dynamic | | | | | Hypervisor | |
|---|---|---|---|---|---|---|---|---|---|---|
| | idH | buildH | netH | accelH | magnFH | rotVecH | proximH | gyrosH | BTdetectH | xFlowH |
| **DroidBox** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **DroidScope** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **TaintDroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **Andrubis** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **SandDroid** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **ApkScan** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **VisualThreat** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Tracedroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **CopperDroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Apk Analyzer** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **ForeSafe** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Mobile Sandbox** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |

# Resilience of dynamic analysis tools

| | Static | | | Dynamic | | | | | Hypervisor | |
|---|---|---|---|---|---|---|---|---|---|---|
| | idH | buildH | netH | accelH | magnFH | rotVecH | proximH | gyrosH | BTdetectH | xFlowH |
| **DroidBox** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **DroidScope** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **TaintDroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **Andrubis** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **SandDroid** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **ApkScan** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **VisualThreat** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Tracedroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **CopperDroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Apk Analyzer** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **ForeSafe** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Mobile Sandbox** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |

All studied services are vulnerable to 5 or more heuristics

# Resilience of dynamic analysis tools

| | Static | | | Dynamic | | | | | Hypervisor | |
|---|---|---|---|---|---|---|---|---|---|---|
| | idH | buildH | netH | accelH | magnFH | rotVecH | proximH | gyrosH | BTdetectH | xFlowH |
| **DroidBox** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **DroidScope** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **TaintDroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **Andrubis** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **SandDroid** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **ApkScan** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **VisualThreat** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Tracedroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **CopperDroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Apk Analyzer** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **ForeSafe** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Mobile Sandbox** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |

These tools failed to infer malicious behavior of the repackaged malware samples

# Resilience of dynamic analysis tools

| | Static | | | Dynamic | | | | | Hypervisor | |
|---|---|---|---|---|---|---|---|---|---|---|
| | idH | buildH | netH | accelH | magnFH | rotVecH | proximH | gyrosH | BTdetectH | xFlowH |
| **DroidBox** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **DroidScope** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **TaintDroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **Andrubis** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **SandDroid** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **ApkScan** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **VisualThreat** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Tracedroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **CopperDroid** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Apk Analyzer** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |
| **ForeSafe** | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| **Mobile Sandbox** | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | *JNI NS* | *JNI NS* |

Only 1 service provides information about VM evasion attempts

# Countermeasures

- Static heuristics
  - Emulator modifications
- Dynamic heuristics
  - Realistic sensor event simulation
- Hypervisor heuristics
  - Accurate binary translation
  - Hardware-assisted virtualization
  - Hybrid application execution

# Summary

- Evaluation of VM evasion to 12 Android dynamic analysis tools
- Only half of the services detected our most trivial heuristics
- No service was resilient to our dynamic and hypervisor heuristics
- Majority of the services failed to detect  repackaged malware
- Only 1 service
  - generated VM evasion attempts
  - was resilient to all our static heuristics

# Rage Against The Virtual Machine: Hindering Dynamic Analysis of Android Malware

Thank you!

**Thanasis Petsas, Giannis Voyatzis, Elias Athanasopoulos, Sotiris Ioannidis,**
{petsas, jvoyatz, elathan, sotiris}@ics.forth.gr
**Michalis Polychronakis,**
mikepo@cs.columbia.edu